Correction TD1 : Organisation et stockage des données

Exerice 1 :

On dispose d'une base de 3 Go constituée d'enregistrements dont la taille moyenne est 3 000 octets.

1. Combien de temps prend la lecture complète de cette base avec un parcours séquentiel ?
2. Combien de temps prend la lecture en effectuant une lecture physique aléatoire pour chaque enregistrement ?

Voir le tableau des caractéristiques des mémoires du support de cours.

1. Le débit d'un disque est de **100 MO/s**. Le temps de lecture de la base est = 3000/100 = **30 secondes.**
2. Nombre d'enregistrements = 3GO/3KO = $3*10^9/3*10^3 = 10^6$ enregistrements. Le temps d'accès est de **10 ms.** Temps de lecture aléatoire = $10^6/10*10^{-3}$ = **10^4 s** = **2 heures 46 minutes 40 secondes**.

Exerice 2 :

Le tableau ci-dessous donne les spécifications partielles d'un disque magnétique. Répondez aux questions suivantes.

1. Quelle est la capacité moyenne d'une piste?, d'un cylindre? d'une surface? du disque?
2. Quel est le temps de latence maximal?
3. Quel est le temps de latence moyen?
4. Combien de temps faut-il pour transmettre le contenu d'une piste et combien de rotations peut effectuer le disque pendant ce temps?

Caractéristique	Valeur
Taille d'un secteur	512 octets
Nbre de plateaux	5
Nbre de têtes	10
Nombre de secteurs	5 335 031 400,00
Nombre de cylindres	10 000
Nombre moyen de secteurs par piste	40 000
Temps de positionnement moyen	10 ms
Vitesse de rotation	7 400 rpm
Déplacement de piste à piste	0,5 ms
Débit moyen	100 Mo / s

1. Capacité d'une piste: 40000 secteurs×512=**20480000 octets ≈ 20 Mo**, Capacité d'un cylindre: 20480000×10 têtes=**204800000 octets≈200Mo**, Capacité d'une surface: 20480000×10000(cylindres=pistes) =**200Go**, Capacité du disque: 200Go*10 = **2 To.**
2. Le temps de latence maximal = une rotation complète. Ici on a 7400/60=0.0081**≈8 ms**.
3. Le temps moyen est celui de ½ tour = **4 ms.**
4. 20 Mo/100 Mo(débit) = 0.2s = **200 ms**. En 200 millisecondes, on peut effectuer 200/8=**25 rotations** (>>1 tour le débit est pénalisant dans ce cas).

Exerice 3 :

Soit un disque de 5 000 cylindres tournant à 12 000 rotations par minute, avec un temps de déplacement entre deux pistes adjacentes égal à 0,002 ms et 500 secteurs de 512 octets par piste.

1. Quel est le temps moyen de lecture d'un bloc de 4,096 octets? Calculer indépendamment le délai de latence, de positionnement et le temps de transfert.

   NB: on suppose que le nombre moyen de déplacements est 1/3 du nombre total de pistes.

1. Délai de latence,: 1/2 rotation. 1 rotation = 60/12000 = 5 ms. Temps moyen de latence=2,5 ms.
2. Temps moyen de positionnement : environ 1/3 du temps total de balayage du disque, soit 0,002*5000(pistes ou cylindres)/3≈3 ms
3. Taux de transfert = 100 Mo/s ➔ Temps de transfert = 4Ko/100Mo~$4*10^{-5}$ s.

Exerice 4 :

Soit la table de schéma suivant:

```
create table Personne (id integer not null,
          nom    varchar(40) not null,
          prenom varchar(40) not null,
          adresse varchar(70),
          dateNaissance date)
```

Cette table contient 300 000 enregistrements, stockés dans des blocs de taille 4 096 octets. Un enregistrement ne peut pas chevaucher deux blocs, et chaque bloc comprend un entête de 200 octets. On ignore l'entête des enregistrements.

1. Donner la taille maximale et la taille minimale d'un enregistrement. On suppose par la suite que tous les enregistrements ont une taille maximale.
2. Quel est le nombre maximal d'enregistrements par bloc?
3. Quelle est la taille du fichier?
4. Quel est le temps moyen de recherche d'un enregistrement si les blocs sont distribués au hasard sur le disque.
5. On suppose que le fichier est trié sur le nom. Quel est le temps d'une recherche dichotomique pour chercher une personne avec un nom donné?

1. Taille minimale,: 4 + 1 + 1 + 0 (null) + 0 (null)=**6**. Taille maximale: 4+41+41+71+8=**165**
2. Espace disponible pour les enregistrements: 4096 - 200 = 3896. Nombre max. d'enregistrements par bloc: ⌊3896/165⌋=**23 blocs.**
3. Taille du fichier: ⌈300000/23⌉=13044 blocs, soit 13044×4096=**53 Mo**
4. Temps d'accès aléatoire selon tableau du cours est de 10 ms pour lire chaque bloc. Au pire des cas, il faut parcourir tout le fichier donc 13044 blocs * 10 ms = **130,440 secondes**.
5. Avec une recherche dichotomique, il faut $\log_2(13044)=13.67 \sim 14$ lectures de blocs, soit **140 ms** au pire des cas.

Exerice 5 :

Reprenons une liste de 12 départements, à lire de gauche à droite et de bas en haut.

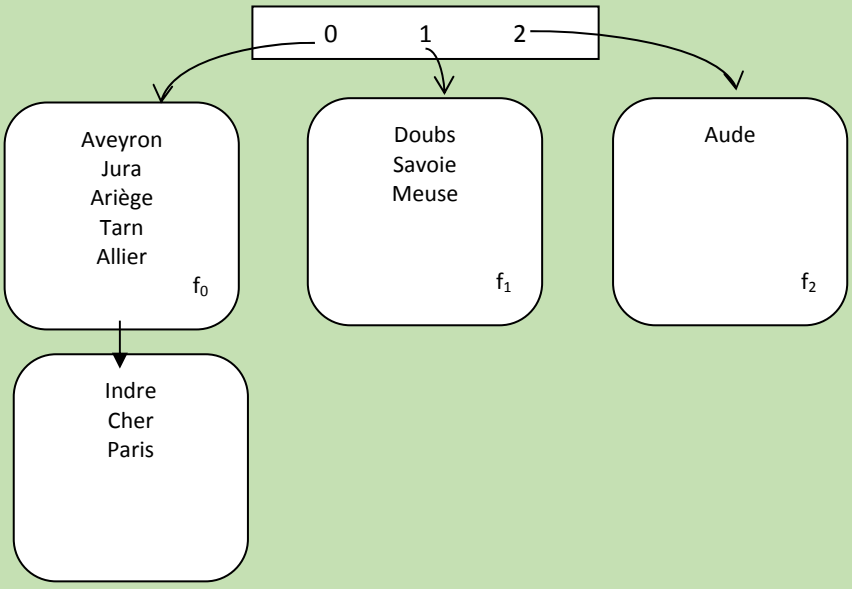
```
3  Allier; 36 Indre; 18 Cher; 75 Paris
39 Jura; 9  Ariège; 81 Tarn; 11 Aude
12 Aveyron; 25 Doubs; 73 Savoie; 55 Meuse;
```

La clé étant le numéro de département et on suppose qu'un bloc contient 5 enregistrements.

1. Proposez une fonction de hachage et le nombre d'entrées du répertoire.
2. Construisez une structure de hachage statique en prenant les enregistrements dans l'ordre de lecture de gauche à droite et de bas en haut.

1. Avec 12 enregistrements, au moins ⌈12/5⌉=**3 blocs** sont nécessaires. La fonction de hachage doit donc fournir le numéro de bloc(0, 1 ou 2). La fonction la plus simple serait c mod 3 où *c* est le code du département.
2. La table de hachage selon l'ordre d'insertion indiqué est comme suit :

```
12 Aveyron (f0); 25 Doubs (f1); 73 Savoie (f1); 55 Meuse (f1); 39
Jura (f0); 9  Ariège (f0); 81 Tarn (f0); 11 Aude (f2); 3  Allier
(f0)débordement; 36 Indre (f0); 18 Cher (f0); 75 Paris (f0)
```



Exerice 6 :

Soit un fichier de données tel que chaque bloc peut contenir 10 enregistrements. On indexe ce fichier avec un niveau d'index, et on suppose qu'un bloc d'index contient 100 entrées *[valeur, adresse]*.

Si *n* est le nombre d'enregistrements, donnez le nombre minimum de blocs en fonction de *n* pour un index dense et un index non-dense.

Le fichier est constitué d'au moins ⌈n/10⌉ blocs. Un index dense contient n entrées et occupe donc ⌈n/100⌉ blocs. Pour un index non-dense il faut seulement ⌈n/10⌉ (même nombre que les blocs) entrées donc ⌈n/1000⌉ blocs.

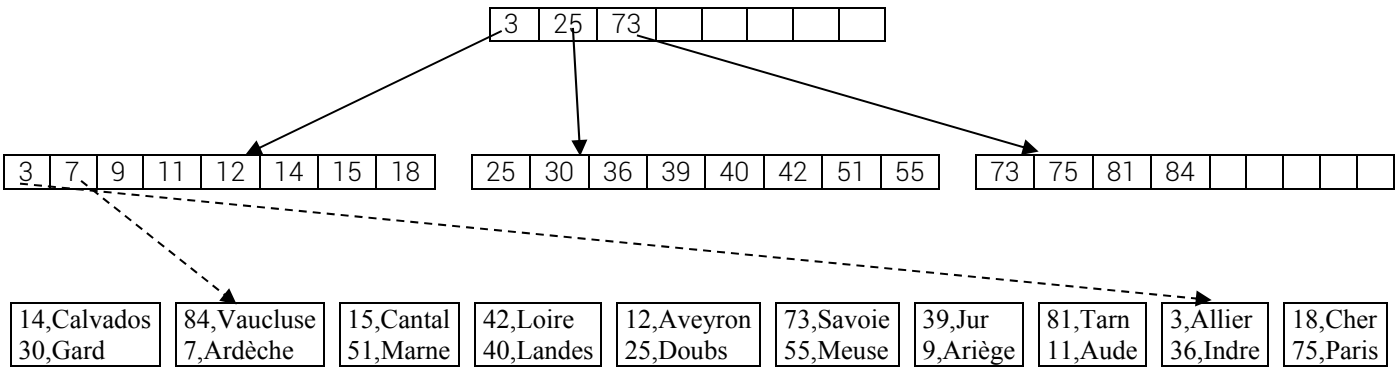
Exerice 7 :

Soit la liste des départements suivants, à lire de gauche à droite et de bas en haut.

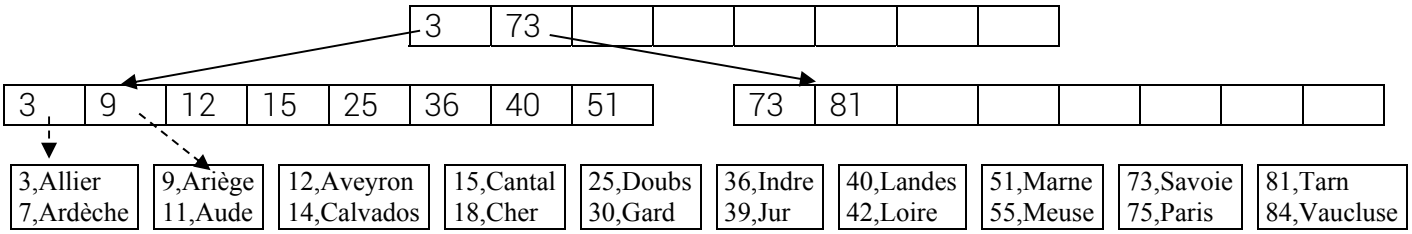
```
3  Allier; 36 Indre; 18 Cher; 75 Paris
39 Jura; 9  Ariège; 81 Tarn; 11 Aude
12 Aveyron; 25 Doubs; 73 Savoie; 55 Meuse;
15 Cantal; 51 Marne; 42 Loire; 40 Landes
14 Calvados; 30 Gard; 84 Vaucluse; 7 Ardèche
```

1. Construire, en prenant comme clé le numéro de département, un index dense à deux niveaux sur le fichier contenant les enregistrements dans l'ordre indiqué ci-dessus, en supposant 2 enregistrements par bloc pour les données, et 8 par bloc pour l'index.
2. Construire un index non-dense sur le fichier trié par numéro, avec les mêmes hypothèses.

1. Index dense : toutes les valeurs distinctes sont présentes dans l'index et 8 entrées par bloc d'index ➔ ⌈20/8⌉ = **3 blocs au premier niveau** ➔ ⌈3/8⌉ = ⌈n/10⌉ et 2 enregistrements par bloc de données ➔ ⌈20/2⌉=**10 blocs de données**.



2. Index non dense : 1 entrée par bloc ➔ 10 entrées ➔ **2 blocs d'index de niveau 1➔ 1 bloc d'index de niveau 2**



Exerice 8 :

Soit les deux tables suivantes:

```
create table R (idR varchar(20) not null,
          primary key (idR));

create table S (idS int not null,
        idR varchar(20) not null,
          primary key (idS),
         foreign key idR references R);
```

Indiquez, pour les ordres SQL suivants, quels index peuvent améliorer les performances ou optimiser la vérification des contraintes « primary key » et « foreign key ».

```
select * from R where idR = 'Bou'
select * from R where idR like 'B%'
select * from R where length(idR) = 3
select * from R where idR like '_ou'
insert into S values (1, 'Bou')
select * from S where idS between 10 and 20
delete from R where idR like 'Z%'
```

1. La première requête est optimisée par l'index sur la clé primaire
2. La seconde est une recherche *par préfixe* sur la clé primaire: idem.
3. La troisième applique une fonction à la clé: l'index ne peut pas être utilisé car le critère de recherche n'est plus une valeur de clé.
4. La quatrième est une recherche *par suffixe*: l'index n'est pas utilisable
5. L'insertion tire parti de l'index sur la clé primaire: il faut vérifier que la valeur de clé n'existe pas déjà. En l'absence d'index il faudrait parcourir toute la table à chaque insertion.
6. La requête par intervalle peut tirer parti de l'index, si l'intervalle est limité.
7. Enfin la destruction dans R implique la vérification qu'il n'existe pas de nuplet référençant dans S, sinon la contrainte d'intégrité serait violée. L'index sur la clé étrangère est ici très utile.